

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

Learning Scala - Seminar 1

Björn Regnell, Christian Söderberg

Dept. of Computer Science
Lund University, Sweden

2012, April 27

Agenda

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ About the course
- ▶ Some highlights of Chapter 1-12
- ▶ Mix interactive plenary explanations and exercises
- ▶ Planning of coming seminars
- ▶ Conclusion and homework

Objectives and qualifications

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ The course is flexible and individualized, depending on qualifications and ambitions.
- ▶ Main objectives
 1. To learn (more about) Scala, including the combination of object orientation + functional programming
 2. To explore if, and if so, how Scala can be used in teaching programming at undergraduate level
- ▶ Minimum qualifications: Object-oriented programming such as Java, Simula, Object Pascal etc.

Course "philosophy"

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Heterogeneous ambitions and learning-goals combined.
- ▶ No questions are stupid questions!
- ▶ We do it together; participants are teachers.
- ▶ The seminars try to make parts of the PINS book come "alive" and to inspire your own experiments.
- ▶ You are expected to study the PINS book between seminars.
- ▶ We reflect on how to teach the studied concepts and their pros and cons in large-scale software development

Flexible Parts overview

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ The course is individualized for each participant to contain one or more of these 3 optional parts depending on the ambition (corresponding to 1.5 - 7.5 ECTS credits):
 - ▶ Part 1: **Scala basics**, Chapters 1-12
1.5 credits, 3 seminars of 2-3 hours each
 - ▶ Part 2: **Advancement in Scala**, Chapters 13-23,
1.5 credits, 2 seminars of 3 hours each
 - ▶ Part 3: **Project work**, Selected portions from Chapters 24-35
2 - 4.5 credits
- Examination: presentation at a seminar, brief report
The project topic is selected freely depending on interest, theoretical or practical studies on anything that is in some way related to Scala.

Seminar planning

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

<i>Part Seminar</i>	<i>Contents</i>	<i>Date</i>
P1 Sem 1	Overview Chapters 1-12	April 27, 13:15-16:00
Exercise	Optional group exercise	May 4, E:2116, 13:15-16:00
P1 Sem 2	Details (taught by course participants) Chapters 1-12	May 29, 13:15-16:00
P1 Sem 3	Invited talks Görel: "The expression problem and Scala" Kris: "Building a Scala library for JaCoP" Jörn: "Scala Actors and Akka" Jacek: "Functional programming in Scala"	June 14, 13:15-15:00
<i>S U M M E R</i>		
P2 Sem 4	Chapters 13-23 taught by course participants	August 29, 13:15-16:00
P2 Sem 5	Chapters 13-23 taught by course participants	September 20, 13:15-16:00
P3 Sem 6	Chapters 24-35 Project presentation by course participants	TBD

Seminar 2: Example of topics

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

At seminar 2, participants will be assigned a time slot and topics to teach, for example:

1. Method overloading, operators and implicit conversion
2. Built-in control structures: if, while, for, exceptions
3. Control abstraction, currying, and by-name-parameters
4. Tail recursion optimization
5. Partially applied functions and special function call forms
6. Inheritance: extending, overriding and final
7. Classes: constructors, auxiliary constructors and access rules
8. Objects: factory objects and method chaining by returning this
9. Scala's class hierarchy and Predef
10. Traits: interfaces, mix-ins and stackable modifications
11. Pedagogical aspects of converting Java assignments to Scala
12. ...

Special topics can be negotiated with Björn.

Participants

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

1. Eva Magnusson, cs lärare
2. Kim Weyns, cs post doc (*)
3. Niklas Fors, cs doktorand
4. Markus Borg, cs doktorand
5. Emma Söderberg, cs doktorand
6. Björn A. Johnsson, cs doktorand
7. Maj Stenmark, cs doktorand (*)
8. Anna Axelsson, cs lärare
9. Per Holm, cs lärare
10. Roger Henriksson, cs lärare
11. Lennart Andersson, cs lärare
12. Jesper Pedersen, cs doktorand
13. Flavius Gruian, cs lärare
14. Sven Gestegard Robertz, cs forskare
15. Alfred Theorin, regler doktorand
16. Görel Hedin, cs lärare
17. Per Andersson, cs lärare
18. Sardar Muhammad Sulaman, cs doktorand
19. Elizabeth Bjarnason, cs doktorand
20. Usman Mazhar Mirza, cs doktorand
21. Martin Höst, cs lärare
22. Mehmet Ali Arslan, cs doktorand (*)
23. Krzysztof Kuchcinski, cs lärare (*)
24. Gustav Cedersjö, cs doktorand
25. Mathias Haage, cs Lärare

(*) Cannot come April 27 but will participate in the course

Today

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Familiarize the participants with the basics of Scala.
- ▶ Big ideas:
 - ▶ Simple programs.
 - ▶ Objects, functions and values.
 - ▶ Higher-order functions.
 - ▶ Introduction to some collection classes.
 - ▶ Simple classes and case classes.
- ▶ If you have already written some Scala programs, you have probably seen everything before.

Pair exercising

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ The one that knows the least about Scala is in control of the keyboard!
- ▶ The one that knows the most about Scala makes an effort to share his/her knowledge in a pedagogical way!
- ▶ If exercises are too easy for you, make them more challenging.
- ▶ If exercises are too difficult don't hesitate to ask.
- ▶ Move now so that you sit together, and tell each other about your scala experiences so far.

Eva Magnusson	+	Gustav Cedersjö
Niklas Fors	+	Markus Borg
Emma Söderberg	+	Alfred Theorin
Björn A. Johnsson	+	Per Andersson
Anna Axelsson	+	Flavius Gruian
Per Holm	+	Jesper Pedersen
Roger Henriksson	+	Lennart Andersson
Sven Gestegard Robertz	+	Usman Mazhar Mirza
Sardar Muhammad Sulaman	+	Elizabeth Bjarnason
Mathias Haage	+	Martin Höst

Quick Intro: What is Scala?

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Started in 2001 at EPFL by Martin Odersky
- ▶ A better Java with cleaned up syntax and relaxed restrictions
- ▶ Combines functional and object-oriented programming
- ▶ Scalable from small scripts to large systems
- ▶ Runs on the Java Virtual Machine
- ▶ Static typing and type inference

How to run Scala

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Programs can be compiled using either `scalac` or `fsc`.
- ▶ Programs can be run as scripts using `scala`.
- ▶ Programs can be run interactively using Scala's Read-Evaluate-Print-Loop (REPL).
- ▶ Programs can also be run interactively using tools such as `Kojo`.
- ▶ Integrated Development Environments for Eclipse, NetBeans and IntelliJ.
- ▶ Also modes for `emacs` (`ensime`), etc.

Example

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

Example

Run the ``hello, world"-program:

- ▶ In the REPL.
- ▶ In Kojo.
- ▶ As a compiled program.
- ▶ As a script.

First exercise: Hello, world!

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
//file hello.scala  
object helloWorld {  
  def main(args: Array[String]) {  
    println("Hello, world!")  
  }  
}
```

Compile with > scalac hello.scala

Run with > scala helloWorld

```
//file myapp.scala  
object helloArgs extends App {  
  println("Hello args: " + args.mkString(", "))  
}
```

compile with > scalac myapp.scala

run with > scala helloArgs hejsan svejsan

In the REPL and in Kojo, no need for object-wrapping, just type:

```
println("Hello, world!")
```

The Scala REPL

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

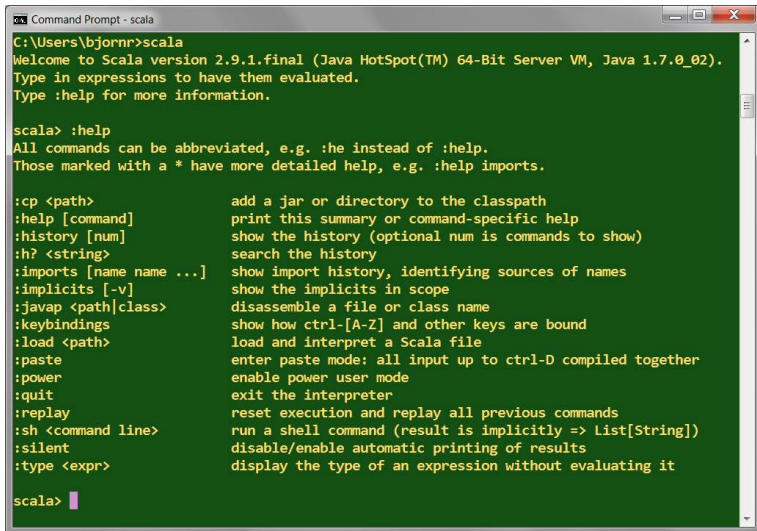
Case class

Object

Values and objects

Collections

Conclusion



```
Command Prompt - scala
C:\Users\bjornr>scala
Welcome to Scala version 2.9.1.final (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_02).
Type in expressions to have them evaluated.
Type :help for more information.

scala> :help
All commands can be abbreviated, e.g. :he instead of :help.
Those marked with a * have more detailed help, e.g. :help imports.

:cp <path>                add a jar or directory to the classpath
:help [command]            print this summary or command-specific help
:history [num]             show the history (optional num is commands to show)
:h? <string>              search the history
:imports [name name ...]  show import history, identifying sources of names
:implicits [-v]           show the implicits in scope
:javap <path|class>       disassemble a file or class name
:keybindings              show how ctrl-[A-Z] and other keys are bound
:load <path>              load and interpret a Scala file
:paste                    enter paste mode: all input up to ctrl-D compiled together
:power                    enable power user mode
:quit                    exit the interpreter
:replay                   reset execution and replay all previous commands
:sh <command line>        run a shell command (result is implicitly => List[String])
:silent                   disable/enable automatic printing of results
:type <expr>              display the type of an expression without evaluating it

scala> 
```

The Kojo IDE for Scala

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

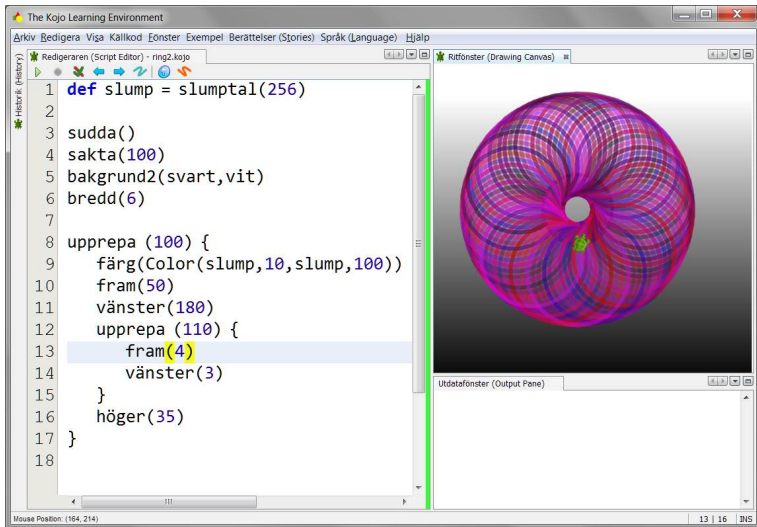
Case class

Object

Values and objects

Collections

Conclusion



Expressions, values, variables & types

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
(1000-90*9.0)/5 + 4    //expression
val x = 42              //x cannot be changed
var y = 41.9            //y can be changed
def z = x + y           //z is evaluated each call
lazy val v = z.round    //evaluation delayed
y = 42.9                //var is updated
println(v)              //v is evaluated
var w: Double = 42.1    //explicit type declaration
var w: Int = 42.1       //error: type mismatch!
```

What is a function?

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

function

A *function* can be *invoked* with a list of arguments to produce a result. A function has a parameter list, a body, and a result type. Functions that are members of a class, trait, or singleton object are called *methods*. Functions defined inside other functions are called *local functions*. Functions with the result type of Unit are called *procedures*. Anonymous functions in source code are called *function literals*. At run time, function literals are instantiated into objects called *function values*.

[PINS Glossary, page 801]

A simple function

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
def add(a:Int, b:Int) = a + b
```

```
add(3,4)
```

Scala function definition syntax

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

The diagram illustrates the syntax of a Scala function definition with the following code and annotations:

```
def max(x: Int, y: Int): Int = {  
    if (x > y)  
        x  
    else  
        y  
}
```

Annotations with arrows pointing to the code:

- "def" starts a function definition (points to `def`)
- function name (points to `max`)
- parameter list in parentheses (points to `(x: Int, y: Int)`)
- function's result type (points to `: Int`)
- equals sign (points to `=`)
- function body in curly braces (points to the block `{ if (x > y) x else y }`)

Scala function literal syntax

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

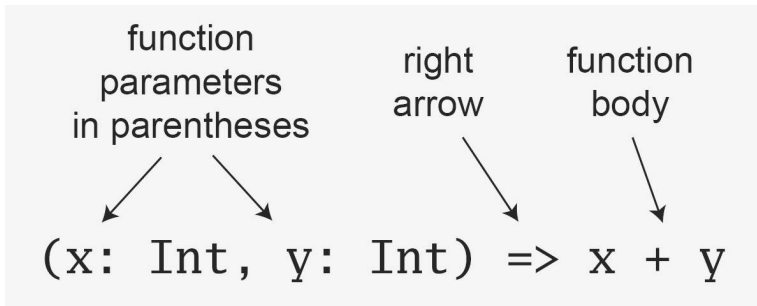
Case class

Object

Values and objects

Collections

Conclusion



[PINS Fig 2.2, page 34]

A simple, anonymous function

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
(a:Int, b:Int) => a + b
```

```
val f = (a:Int, b:Int) => a + b
```

```
f(3,4)
```

What is a closure?

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

A closure is a function that refers to non-local names that are in scope at the point of declaration, and where the binding to actual values are made at the point of evaluation.

```
var a = 43
def f(x:Int) = x + a //access to non-local name
println(f(-1))      //f is closed over a bound to 43
a = 41
println(f(1))       //f is closed over a bound to 41
```

Class

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
class Point(initX:Double, initY:Double) {  
    var x = initX  
    var y = initY  
    override def toString = "Point("+x+", "+y+")"  
}  
  
val p = new Point(0,0)  
println(p)  
p.x = 10    // OK, p is a val, but p.x is a var  
p.y = 20  
println(p)  
  
//p.initX = 3 //not allowed!!
```

Case class

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

A case class is a class that has several things ready-made, including a nice toString and a factory object so that we do not need to write "new". Case classes are powerful tools for pattern-matching.

```
case class Point(x:Double, y:Double)
val p = Point(10,20)
println(p)
```

//p.x = 3 //not allowed, x is a val!!

Singleton object

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

If you only need one instance you do not need to declare and instantiate a class, you can make a singleton object directly:

```
object myObject {  
    val myAttribute = 42  
}
```

```
println(myObject.myAttribute)
```

Values are objects, operations are methods

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

1 + 2

Values are objects, operations are methods

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

1 + 2

- *Every value is an object* -- we don't have to differentiate between primitive types and reference types.

Values are objects, operations are methods

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

$1 + 2$

- ▶ *Every value is an object* -- we don't have to differentiate between primitive types and reference types.
- ▶ *Every operation is a method call*, the expression $1 + 2$ is just shorthand for $1.+(2)$

Values are objects, operations are methods

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

1 + 2

- ▶ *Every value is an object* -- we don't have to differentiate between primitive types and reference types.
- ▶ *Every operation is a method call*, the expression `1 + 2` is just shorthand for `1.+(2)`
- ▶ The syntax rules allows us to skip dots, parentheses and semicolons in many places. We can also sometimes chose between using `{}` and `()`.

Values are objects, operations are methods

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

1 + 2

- ▶ *Every value is an object* -- we don't have to differentiate between primitive types and reference types.
- ▶ *Every operation is a method call*, the expression `1 + 2` is just shorthand for `1.+(2)`
- ▶ The syntax rules allows us to skip dots, parentheses and semicolons in many places. We can also sometimes chose between using `{}` and `()`.
- ▶ The `==`-operator checks values (using `equals`), not references.

Int-values and for

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- Scala has a class `Int`, with lots of methods, such as:

```
val indianapi = math.Pi.toInt
```

```
val biggest = a max b
```

The values are compiled into the same 32-bit signed int-values that Java has.

Int-values and for

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- Scala has a class `Int`, with lots of methods, such as:

```
val indianaPi = math.Pi.toInt  
val biggest = a max b
```

The values are compiled into the same 32-bit signed int-values that Java has.

- One noteworthy method on `Int`s is to:

```
val scale = 0 to 11
```

It returns a `Range`-object, which is a kind of collection whose values are generated 'on demand'.

Int-values and for

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala has a class `Int`, with lots of methods, such as:

```
val indianapi = math.Pi.toInt  
val biggest = a max b
```

The values are compiled into the same 32-bit signed int-values that Java has.

- ▶ One noteworthy method on `Int`s is to:

```
val scale = 0 to 11
```

It returns a `Range`-object, which is a kind of collection whose values are generated 'on demand'.

- ▶ We can use a `for`-statement to loop through a `Range`:

```
for (k <- 1 to 10)  
  println(k)
```

Values are all around

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ The if-statement has a value:

```
val smallest = if (a < b) a else b
```

Values are all around

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ The if-statement has a value:

```
val smallest = if (a < b) a else b
```

- ▶ Blocks have values (their last calculated value):

```
val gauss = {  
  var sum = 0  
  for (term <- 1 to 100)  
    sum += term  
  sum  
}
```

Values are all around

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ The if-statement has a value:

```
val smallest = if (a < b) a else b
```

- ▶ Blocks have values (their last calculated value):

```
val gauss = {  
  var sum = 0  
  for (term <- 1 to 100)  
    sum += term  
  sum  
}
```

- ▶ The Scala counterpart to Java's void is called Unit, and its only value is (). A block with a closing Unit-statement (such as println) has the value ().

Tuples

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- In Scala we can return several values from a function, using a tuple.

Tuples

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ In Scala we can return several values from a function, using a tuple.
- ▶ Tuples can be of any arity up to 22 -- we can write a tuple with one Int, one Boolean, and one String as:

```
(42, true, "hello, world")
```

Tuples

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ In Scala we can return several values from a function, using a tuple.
- ▶ Tuples can be of any arity up to 22 -- we can write a tuple with one Int, one Boolean, and one String as:

```
(42, true, "hello, world")
```

Example

Write a function which takes three integers, and returns a 3-tuple with the values in ascending order.

Solution

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
object MinMidMax extends App {  
  def minmidmax(a: Int, b: Int, c: Int) = {  
    val smallest = a min b min c  
    val biggest = a max b max c  
    (smallest,  
     a + b + c - smallest - biggest,  
     biggest)  
  }  
  val (a, b, c) = minmidmax(5, 2, 4)  
  println("%d, %d, %d".format(a, b, c))  
}
```

Functions are values

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- Functions are actually objects with an `apply`-method.

Functions are values

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Functions are actually objects with an `apply`-method.
- ▶ Using some syntactic sugaring, the compiler allows us to write

```
square(a) + square(b)
```

instead of

```
square.apply(a).+(square.apply(b))
```

Functions are values

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Functions are actually objects with an `apply`-method.
- ▶ Using some syntactic sugaring, the compiler allows us to write

```
square(a) + square(b)
```

instead of

```
square.apply(a).+(square.apply(b))
```

- ▶ Scala converts methods into functions when needed, so we can use them almost interchangeably.

Functions are values

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Functions are actually objects with an `apply`-method.
- ▶ Using some syntactic sugaring, the compiler allows us to write

```
square(a) + square(b)
```

instead of

```
square.apply(a).+(square.apply(b))
```

- ▶ Scala converts methods into functions when needed, so we can use them almost interchangeably.
- ▶ Since they're values, we can send functions/methods as parameters to other functions.

Functions are values

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Functions are actually objects with an `apply`-method.
- ▶ Using some syntactic sugaring, the compiler allows us to write

```
square(a) + square(b)
```

instead of

```
square.apply(a).+(square.apply(b))
```

- ▶ Scala converts methods into functions when needed, so we can use them almost interchangeably.
- ▶ Since they're values, we can send functions/methods as parameters to other functions.
- ▶ We can also return a function as a value from a function.

Example

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

Example

Write a function which tabulates another function in a given interval.

Solution

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
object Tabulate extends App {  
  def tabulate(f: Double => Double,  
              min: Double,  
              max: Double) =  
    (min to max by 0.1).foreach {  
      x => println("%10.4f %10.4f".format(x, f(x)))  
    }  
  
  def square(x: Double) = x * x  
  println("Table for the square function:")  
  tabulate(square, 0, 1)  
  println("Table for the cube function:")  
  tabulate(x => x * x * x, 0, 1)  
  println("Table for the square root function:")  
  tabulate(math.sqrt, 0, 1)  
}
```

Scala's collections

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala's standard collections come in three flavors:

Scala's collections

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- Scala's standard collections come in three flavors:
 - immutable (default)

Scala's collections

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- Scala's standard collections come in three flavors:
 - immutable (default)
 - mutable

Scala's collections

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- Scala's standard collections come in three flavors:
 - immutable (default)
 - mutable
 - parallel

Scala's collections

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala's standard collections come in three flavors:
 - ▶ immutable (default)
 - ▶ mutable
 - ▶ parallel
- ▶ Each flavor has sets, sequences, and maps -- and sequences are divided into indexed sequences and linear sequences.

Scala's collections

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala's standard collections come in three flavors:
 - ▶ immutable (default)
 - ▶ mutable
 - ▶ parallel
- ▶ Each flavor has sets, sequences, and maps -- and sequences are divided into indexed sequences and linear sequences.
- ▶ Instead of looping through our collections, we often send functions ('closures') as parameters to methods on the collections.

Scala's collections

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala's standard collections come in three flavors:
 - ▶ immutable (default)
 - ▶ mutable
 - ▶ parallel
- ▶ Each flavor has sets, sequences, and maps -- and sequences are divided into indexed sequences and linear sequences.
- ▶ Instead of looping through our collections, we often send functions ('closures') as parameters to methods on the collections.
- ▶ These collection-methods can be seen as control structures (and they often replace `while` and `for`).

Scala's collections

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala's standard collections come in three flavors:
 - ▶ immutable (default)
 - ▶ mutable
 - ▶ parallel
- ▶ Each flavor has sets, sequences, and maps -- and sequences are divided into indexed sequences and linear sequences.
- ▶ Instead of looping through our collections, we often send functions ('closures') as parameters to methods on the collections.
- ▶ These collection-methods can be seen as control structures (and they often replace `while` and `for`).
- ▶ By using collections and higher order functions we can solve many problems surprisingly easily.

scala.collection.immutable

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion



scala.collection.mutable

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

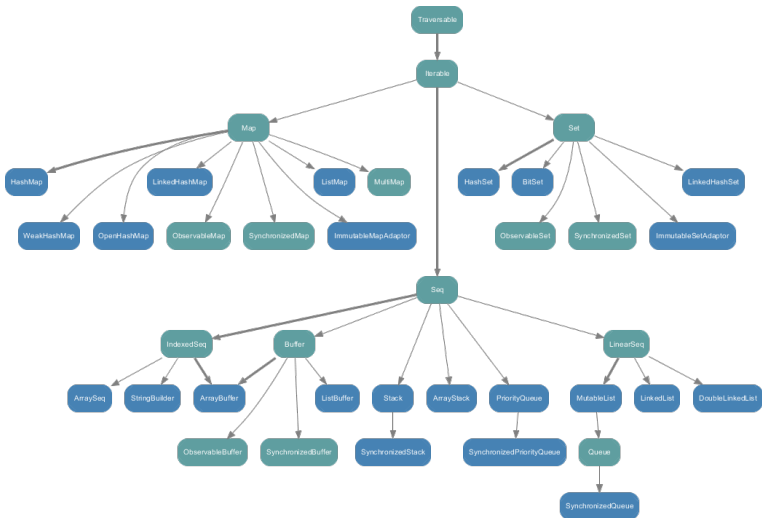
Case class

Object

Values and objects

Collections

Conclusion



Collections and higher-order-functions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- To print every value in a sequence of numbers, we can call the `foreach`-method on the sequence, and as a parameter send a function which prints a value:

```
(1 to 10).foreach((p: Int) => println(p))
```

Collections and higher-order-functions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ To print every value in a sequence of numbers, we can call the `foreach`-method on the sequence, and as a parameter send a function which prints a value:

```
(1 to 10).foreach((p: Int) => println(p))
```

- ▶ The type of `p` can be inferred by the compiler:

```
(1 to 10).foreach(p => println(p))
```

Collections and higher-order-functions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ To print every value in a sequence of numbers, we can call the `foreach`-method on the sequence, and as a parameter send a function which prints a value:

```
(1 to 10).foreach((p: Int) => println(p))
```

- ▶ The type of `p` can be inferred by the compiler:

```
(1 to 10).foreach(p => println(p))
```

- ▶ We can replace the occurrence of one parameter with `_` as in:

```
(1 to 10).foreach(println(_))
```

Collections and higher-order-functions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ To print every value in a sequence of numbers, we can call the `foreach`-method on the sequence, and as a parameter send a function which prints a value:

```
(1 to 10).foreach((p: Int) => println(p))
```

- ▶ The type of `p` can be inferred by the compiler:

```
(1 to 10).foreach(p => println(p))
```

- ▶ We can replace the occurrence of one parameter with `_` as in:

```
(1 to 10).foreach(println(_))
```

- ▶ This can be simplified even further, into:

```
(1 to 10).foreach(println)
```

Collections and closures

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

Example

Calculate $1 + 2 + \dots + 100$ using foreach and a closure.

Collections and closures

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

Example

Calculate $1 + 2 + \dots + 100$ using foreach and a closure.

Example

Assuming all command line arguments are integers, print those which are bigger than 42.

Solution

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
object Sum extends App {  
  var sum = 0  
  (1 to 100).map(k => k * k).foreach(sum += _)  
  println(sum)  
}  
  
object BigNumbers extends App {  
  val bigEnough = args.map(_.toInt).filter(_ >= 42).reverse  
  println(bigEnough.mkString(", "))  
}
```

Some methods on a Seq[A]

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ `foreach(f: A => Unit): Unit` -- applies `f` to every value in the sequence (returning nothing).

Some methods on a Seq[A]

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ `foreach(f: A => Unit): Unit` -- applies `f` to every value in the sequence (returning nothing).
- ▶ `map[B](f: A => B): Seq[B]` -- takes a sequence $a_1, a_2, \dots$ and returns $f(a_1), f(a_2), \dots$

Some methods on a Seq[A]

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ `foreach(f: A => Unit): Unit` -- applies `f` to every value in the sequence (returning nothing).
- ▶ `map[B](f: A => B): Seq[B]` -- takes a sequence $a_1, a_2, \dots$ and returns $f(a_1), f(a_2), \dots$.
- ▶ `filter(p: A => Boolean): Seq[A]` -- gives a sequence with those values in the original sequence for which `p` returns true.

Some methods on a Seq[A]

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ `foreach(f: A => Unit): Unit` -- applies `f` to every value in the sequence (returning nothing).
- ▶ `map[B](f: A => B): Seq[B]` -- takes a sequence $a_1, a_2, \dots$ and returns $f(a_1), f(a_2), \dots$.
- ▶ `filter(p: A => Boolean): Seq[A]` -- gives a sequence with those values in the original sequence for which `p` returns true.
- ▶ `take(n: Int): Seq[A]` -- gives the first n values of the sequence.

Some methods on a Seq[A]

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ `foreach(f: A => Unit): Unit` -- applies `f` to every value in the sequence (returning nothing).
- ▶ `map[B](f: A => B): Seq[B]` -- takes a sequence $a_1, a_2, \dots$ and returns $f(a_1), f(a_2), \dots$.
- ▶ `filter(p: A => Boolean): Seq[A]` -- gives a sequence with those values in the original sequence for which `p` returns true.
- ▶ `take(n: Int): Seq[A]` -- gives the first n values of the sequence.
- ▶ `sortWith(lt: (A, A)):` `Seq[A]` -- gives a sorted sequence, where the elements are compared using the function `lt`.

Some methods on a Seq[A]

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ `foreach(f: A => Unit): Unit` -- applies `f` to every value in the sequence (returning nothing).
- ▶ `map[B](f: A => B): Seq[B]` -- takes a sequence $a_1, a_2, \dots$ and returns $f(a_1), f(a_2), \dots$.
- ▶ `filter(p: A => Boolean): Seq[A]` -- gives a sequence with those values in the original sequence for which `p` returns true.
- ▶ `take(n: Int): Seq[A]` -- gives the first n values of the sequence.
- ▶ `sortWith(lt: (A, A)):` `Seq[A]` -- gives a sorted sequence, where the elements are compared using the function `lt`.
- ▶ `sum: A` -- gives the sum of all the values of the sequence (works only for types which can be converted to numeric values).

Strings and implicit conversions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala has its own classes for Int, Double, etc, but it uses Java's String-class.

Strings and implicit conversions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala has its own classes for Int, Double, etc, but it uses Java's String-class.
- ▶ So when we write `s.charAt(k)` we are calling the charAt-method defined in the `java.lang.String`-class.

Strings and implicit conversions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala has its own classes for `Int`, `Double`, etc, but it uses Java's `String`-class.
- ▶ So when we write `s.charAt(k)` we are calling the `charAt`-method defined in the `java.lang.String`-class.
- ▶ Scala also define a class `StringOps`, with additional methods, among those all `Seq`-methods (a string is, after all, a sequence of `Char:s`).

Strings and implicit conversions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala has its own classes for `Int`, `Double`, etc, but it uses Java's `String`-class.
- ▶ So when we write `s.charAt(k)` we are calling the `charAt`-method defined in the `java.lang.String`-class.
- ▶ Scala also define a class `StringOps`, with additional methods, among those all `Seq`-methods (a string is, after all, a sequence of `Char:s`).
- ▶ If we try to call a `Seq`-method, such as `reverse`, on a `String`-value, Scala will automagically wrap the string in a `StringOps`-object.

Strings and implicit conversions

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Scala has its own classes for `Int`, `Double`, etc, but it uses Java's `String`-class.
- ▶ So when we write `s.charAt(k)` we are calling the `charAt`-method defined in the `java.lang.String`-class.
- ▶ Scala also define a class `StringOps`, with additional methods, among those all `Seq`-methods (a string is, after all, a sequence of `Char:s`).
- ▶ If we try to call a `Seq`-method, such as `reverse`, on a `String`-value, Scala will automagically wrap the string in a `StringOps`-object.
- ▶ This technique is called *implicit conversion*, and it's used extensively in Scala's standard library.

Example

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

Example

Write a program which prints, in alphabetical order, all palindromes given on the command line.

Solution

Learning Scala

Seminar 1

About the
Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

```
object Palindromes extends App {  
  def isPalindrome(s: String) = s.reverse == s  
  args  
    .filter(isPalindrome)  
    .sortWith(_ < _)  
    .foreach(println)  
}
```

Conclusion & Homework

Learning Scala

Seminar 1

About the Course

Seminar plan

Quick Intro

Hello, world!

val, var, type

Functions

Closure

Class

Case class

Object

Values and objects

Collections

Conclusion

- ▶ Skim the PINS book Chapters 1-12
- ▶ Check out ->Resources on the course web page:
http://cs.lth.se/english/course/learning_scala/
- ▶ Familiarize yourself with ScalaDoc
- ▶ Do selected ->Exercises on the course web page
- ▶ You will be assigned a time slot and an area from Chapters 1-12 that you will teach at Seminar 2.
- ▶ Swaps of areas among participants ok if mutually agreed.